

C O M P U T A T I O N A L R E S E A R C H D I V I S I O N

Multicore Optimization of Sparse Matrix Vector Multiplication

Leonid Oliker^{1,2}

Samuel Williams^{1,2}, Richard Vuduc³,
John Shalf¹, Katherine Yelick^{1,2}, James Demmel^{1,2}

¹Lawrence Berkeley National Laboratory

²University of California Berkeley

³Georgia Institute of Technology

Overview

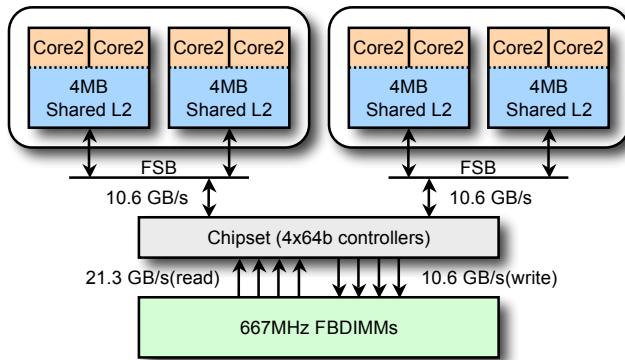


- ❖ Multicore is the architectural paradigm for the next decade
 - CMP offers best tradeoffs: performance, energy, and reliability
- ❖ Diversity of CMP raise difficult questions between design choices:
 - Performance, application suitability, software optimization
- ❖ Examined Sparse Matrix Vector Multiplication (SpMV) kernel
 - Important HPC kernel, memory intensive, challenging for multicore
- ❖ Present two autotuned threaded implementations:
 - Pthread, cache-based implementation
 - Cell local store-based implementation
- ❖ Benchmarked performance across 4 diverse multicore architectures
 - Intel Xeon Clovertown, AMD Opteron, Sun Niagara2, STI Cell
- ❖ Compare performance with leading MPI implementation (PETSc)

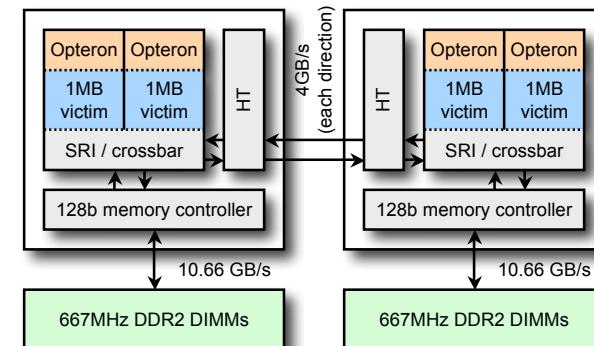
Multicore SMP Systems



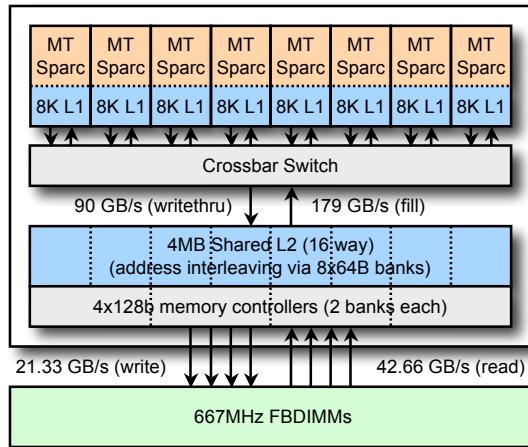
Intel Clovertown



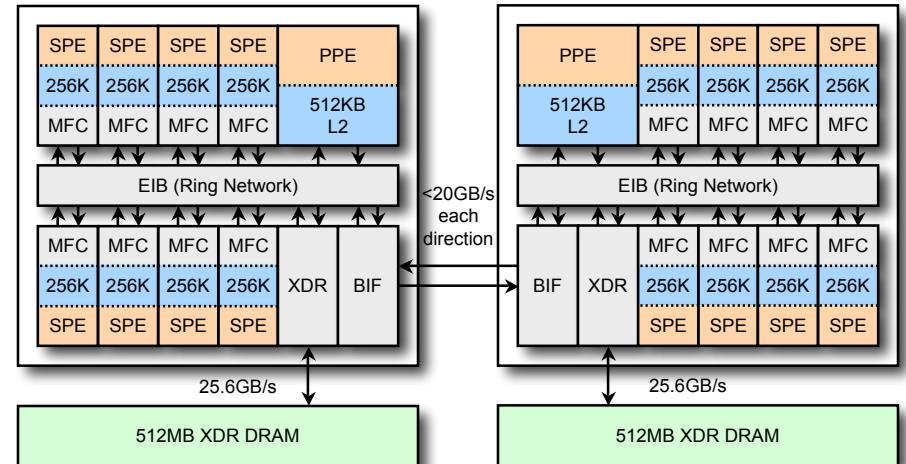
AMD Opteron



Sun Niagara2 (Huron)



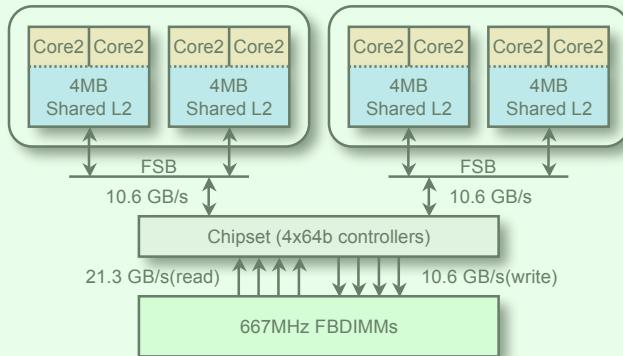
IBM QS20 Cell Blade



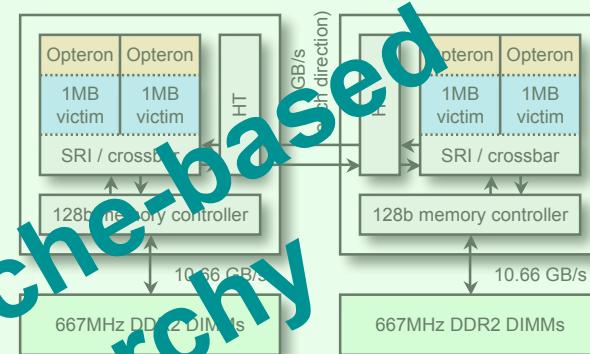
Multicore SMP Systems (memory hierarchy)



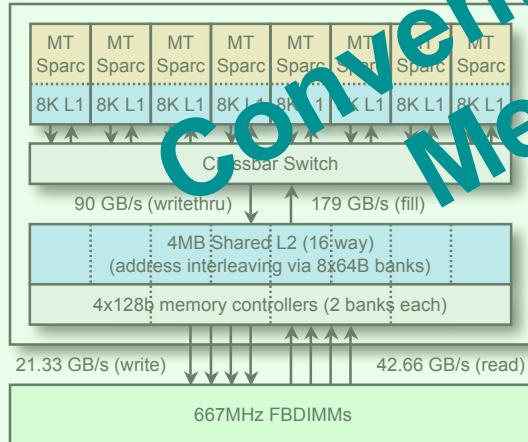
Intel Clovertown



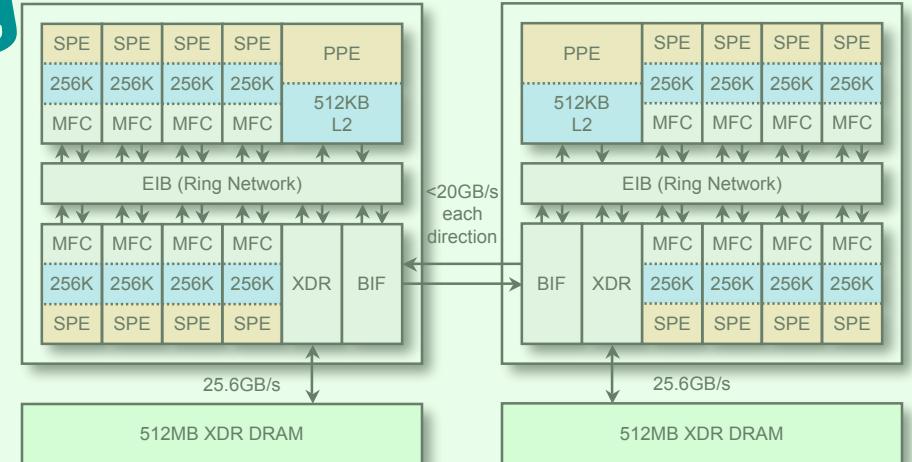
AMD Opteron



Sun Niagara2 (Huron)



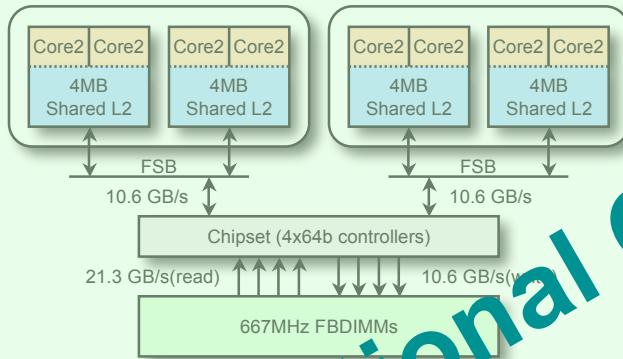
IBM QS20 Cell Blade



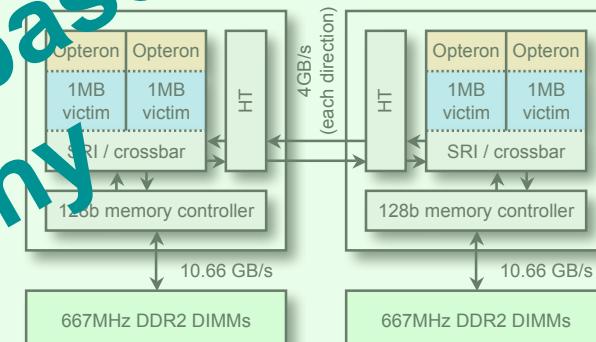
Multicore SMP Systems (memory hierarchy)



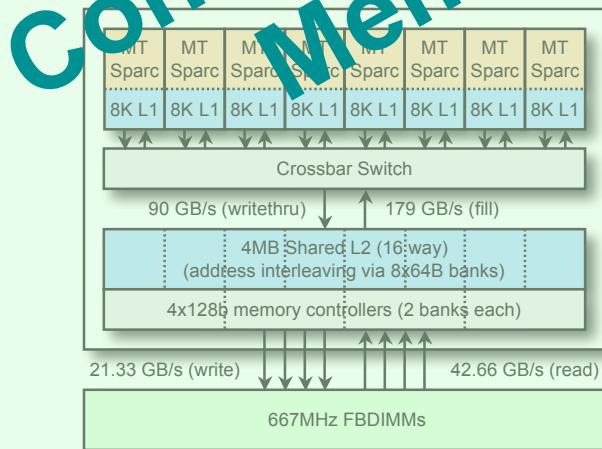
Intel Clovertown



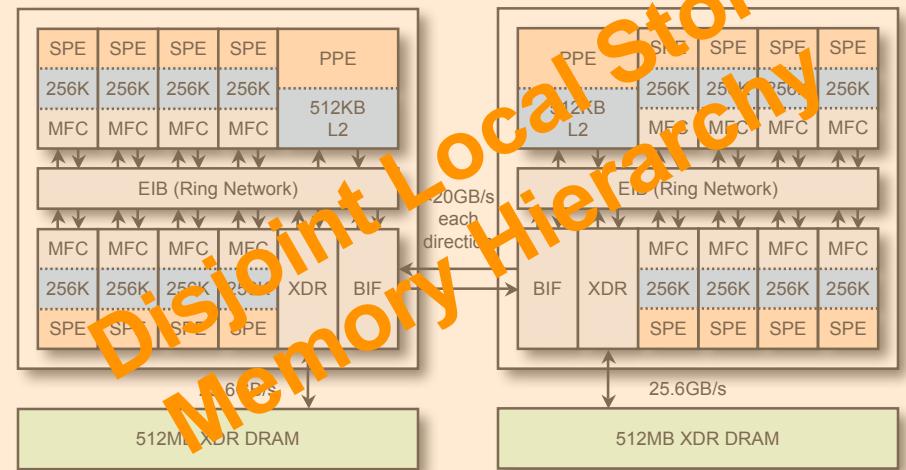
AMD Opteron



Sun Niagara2 (Huron)



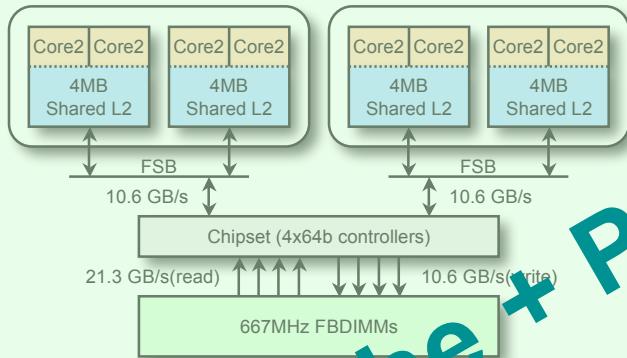
IBM QS20 Cell Blade



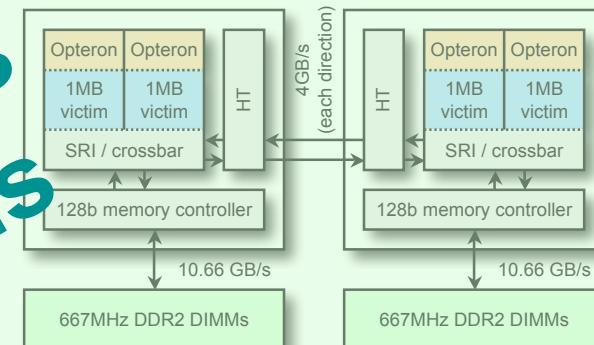
Multicore SMP Systems (memory hierarchy)



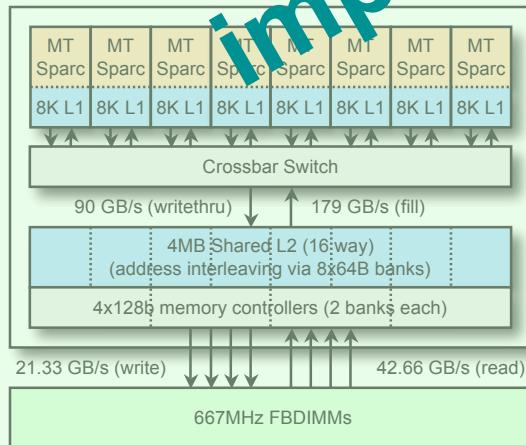
Intel Clovertown



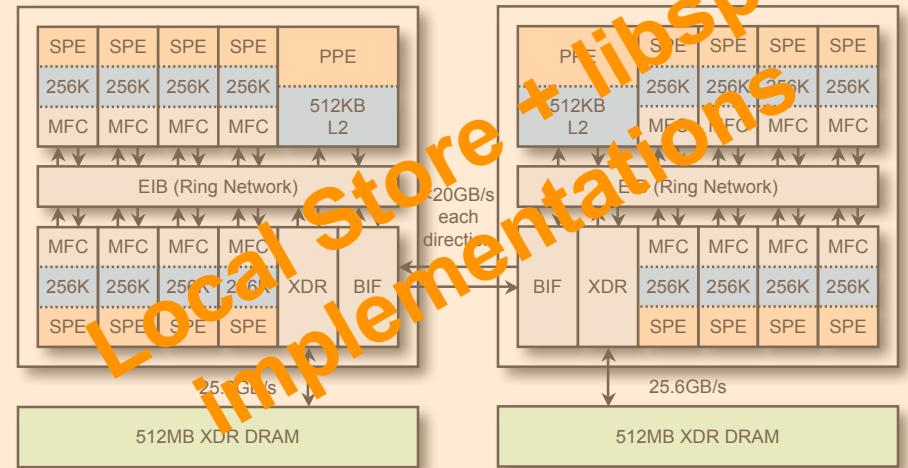
AMD Opteron



Sun Niagara2 (Hudson)



IBM QS20 Cell Blade

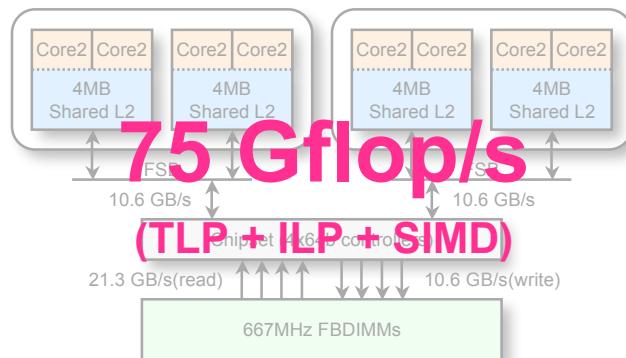


Multicore SMP Systems

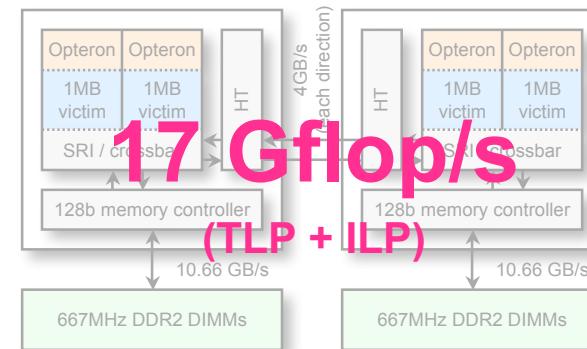
(peak flops)



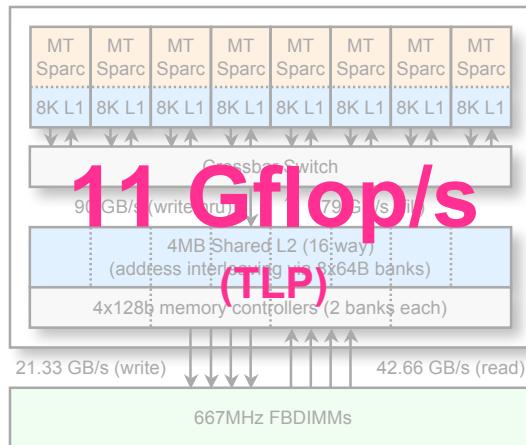
Intel Clovertown



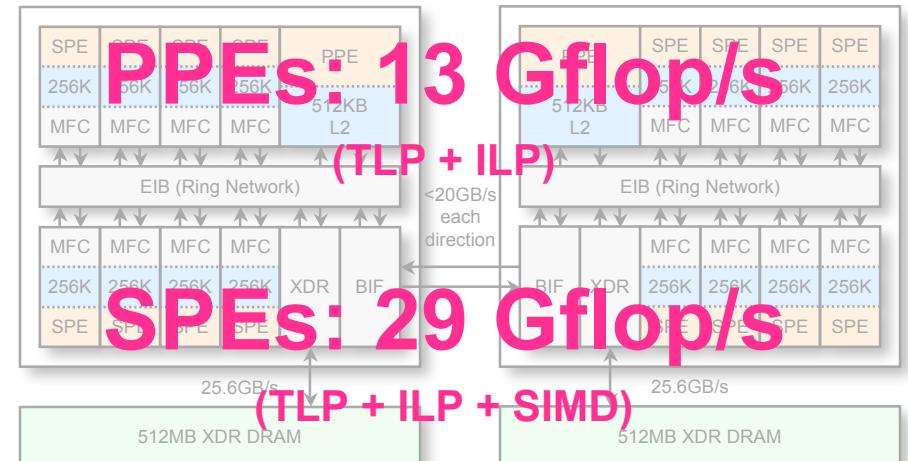
AMD Opteron



Sun Niagara2 (Huron)



IBM QS20 Cell Blade

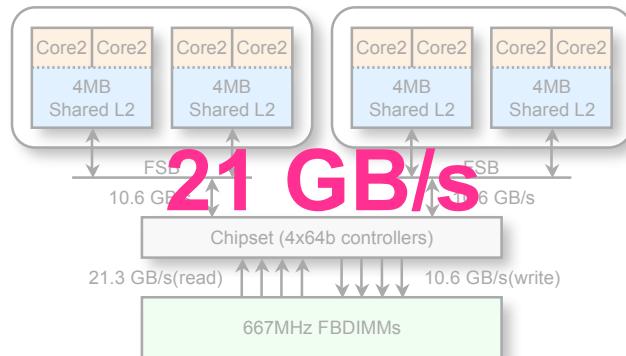


Multicore SMP Systems

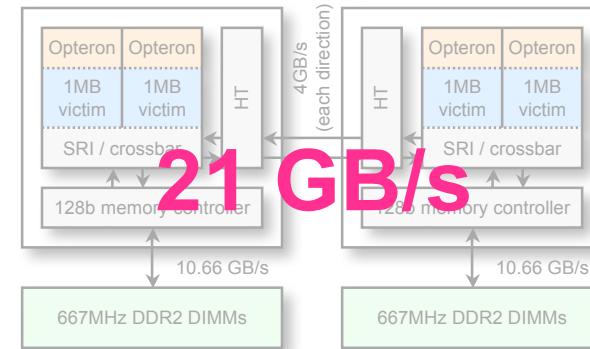
(peak DRAM bandwidth)



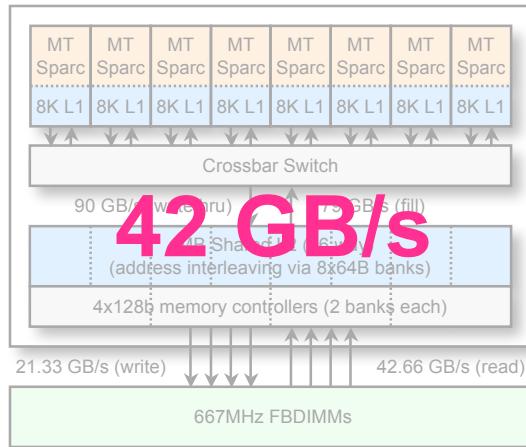
Intel Clovertown



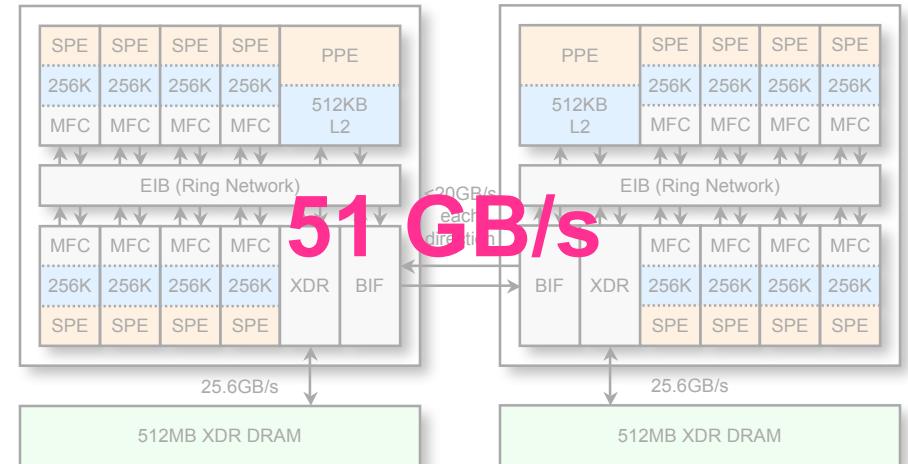
AMD Opteron



Sun Niagara2 (Huron)



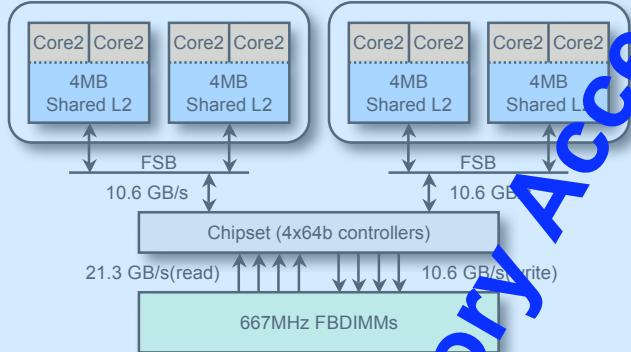
IBM QS20 Cell Blade



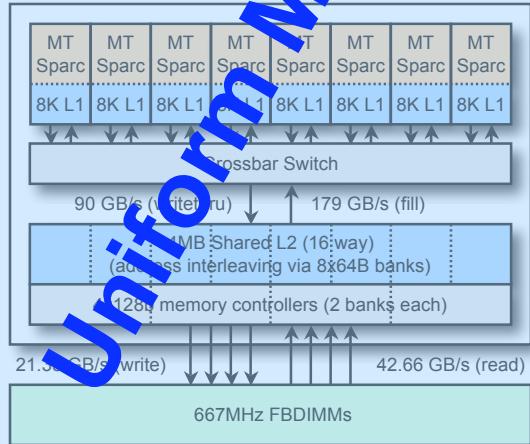
Multicore SMP Systems



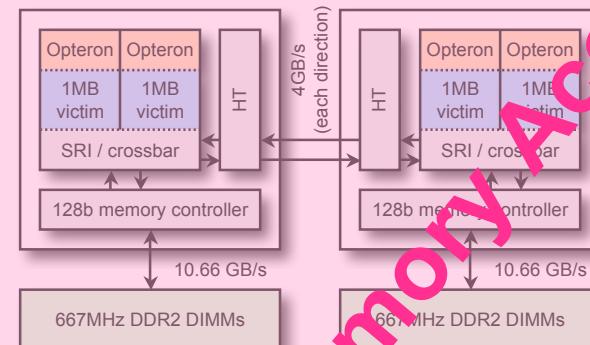
Intel Clovertown



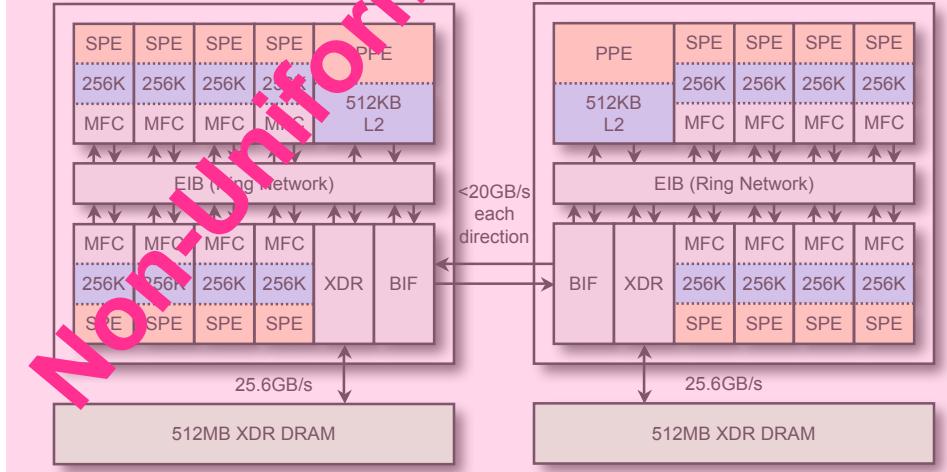
Sun Niagara2 (Huron)



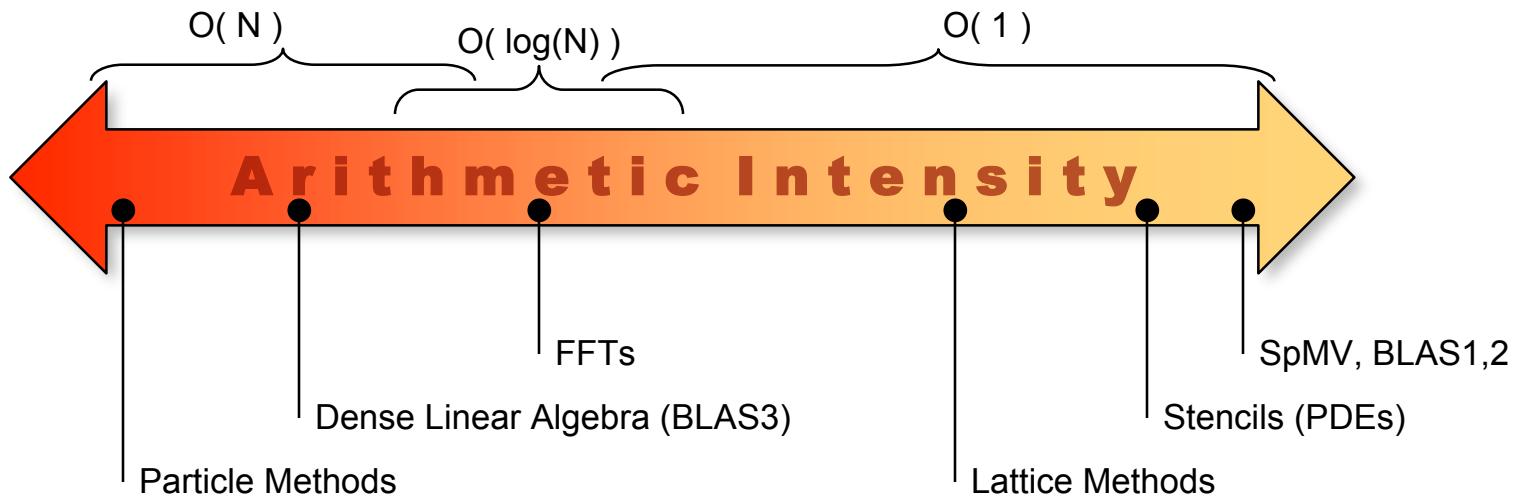
AMD Opteron



IBM QS20 Cell Blade



Arithmetic Intensity



- ❖ Many HPC kernels have an arithmetic intensity that scales with problem size (increasing temporal locality)
- ❖ But there are many important and interesting kernels that don't
- ❖ Low arithmetic intensity kernels are likely to be memory bound
- ❖ High arithmetic intensity kernels are likely to be processor bound

Sparse Matrix Vector Multiplication



- ❖ Sparse Matrix
 - Most entries are 0.0
 - Performance advantage in only storing/operating on the nonzeros
 - Requires significant meta data
- ❖ Evaluate $y = Ax$
 - A is a sparse matrix
 - x & y are dense vectors
- ❖ Challenges
 - Difficult to exploit ILP (inhibits superscalars),
 - Difficult to exploit DLP(inhibits SIMD)
 - Irregular memory access to source vector
 - Difficult to load balance
 - **Very low computational intensity (often >6 bytes/flop)**

$$\begin{bmatrix} \text{Sparse Matrix} \\ A \end{bmatrix} \times \begin{bmatrix} \text{Dense Vector} \\ x \end{bmatrix} = \begin{bmatrix} \text{Dense Vector} \\ y \end{bmatrix}$$
A diagram illustrating the operation of Sparse Matrix Vector Multiplication (SpMV). On the left, a large square matrix labeled 'A' is shown with a sparse pattern of gray squares on a white background. To its right is a vertical vector labeled 'x'. Between them is a multiplication symbol ('×'). To the right of the multiplication symbol is another vertical vector labeled 'y'. This visualizes the equation $Ax = y$.

Dataset (Matrices)

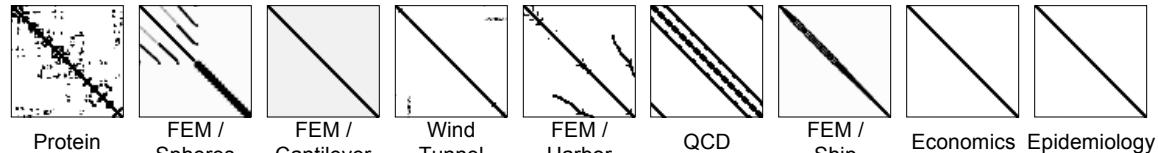


2K x 2K Dense matrix
stored in sparse format



Dense

Well Structured
(sorted by nonzeros/row)



Protein

FEM / Spheres

FEM /
Cantilever

Wind
Tunnel

FEM /
Harbor

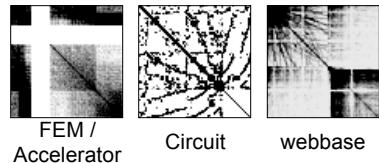
QCD

FEM /
Ship

Economics

Epidemiology

Poorly Structured
hodgepodge



FEM /
Accelerator

Circuit

webbase

Extreme Aspect Ratio
(linear programming)



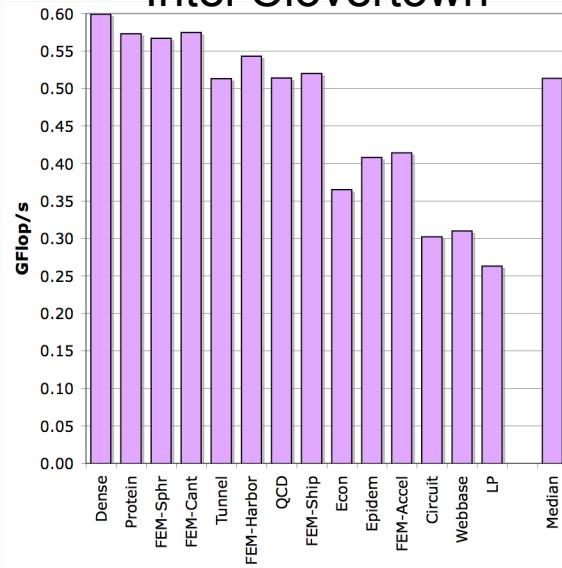
LP

- ❖ Pruned original SPARSITY suite down to 14
- ❖ Large enough not to fit in cache
- ❖ Subdivided them into 4 categories
- ❖ Rank ranges from 2K to 1M

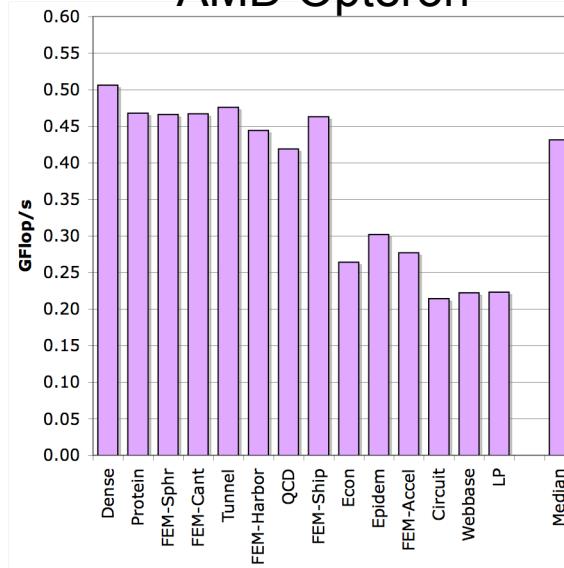
Naïve Serial Implementation



Intel Clovertown

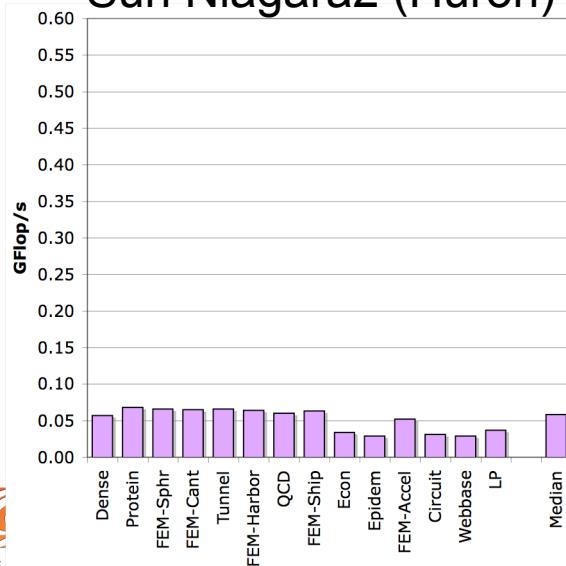


AMD Opteron

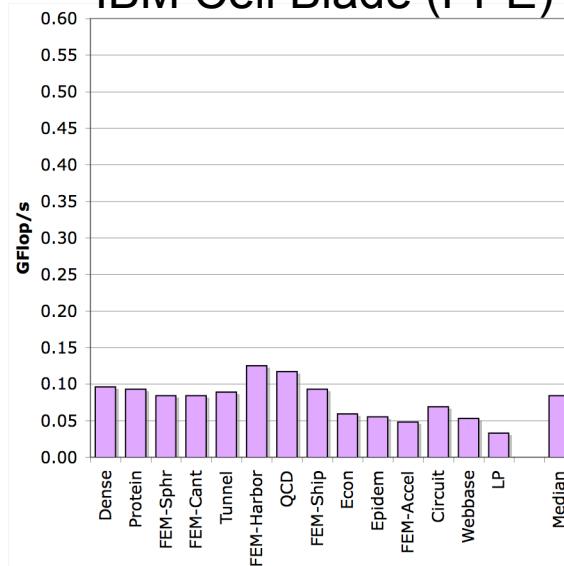


- ❖ Vanilla C implementation
- ❖ Matrix stored in CSR (compressed sparse row)
- ❖ Explored compiler options, but only the best is presented here
- ❖ x86 core delivers > 10x the performance of a Niagara2 thread

Sun Niagara2 (Huron)



IBM Cell Blade (PPE)





C O M P U T A T I O N A L R E S E A R C H D I V I S I O N

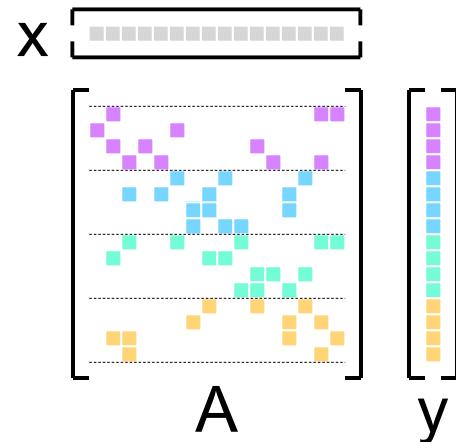
Pthread Implementation

- ❖ Optimized for multicore/threading
- ❖ Variety of shared memory programming models are acceptable(not just Pthreads)
- ❖ More colors = more optimizations = more work

Parallelization



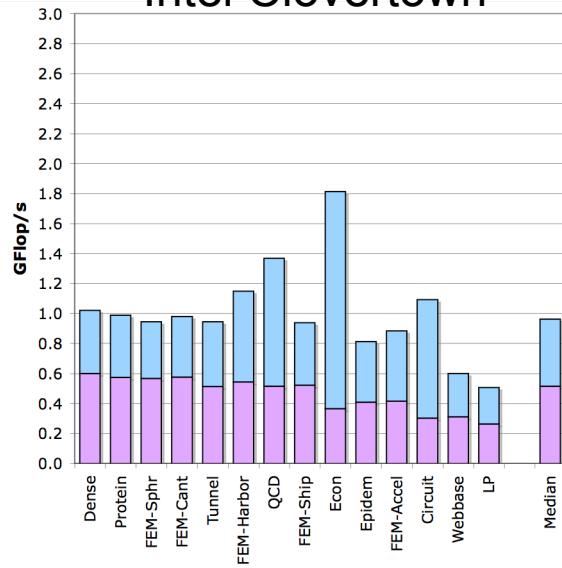
- ❖ Matrix partitioned by rows and balanced by the number of nonzeros
- ❖ SPMD like approach
- ❖ A barrier() is called before and after the SpMV kernel
- ❖ Each sub matrix stored separately in CSR
- ❖ Load balancing can be challenging



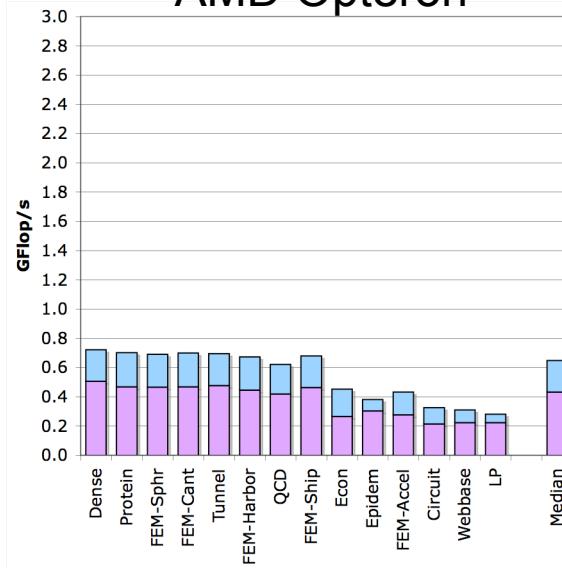
Naïve Parallel Implementation



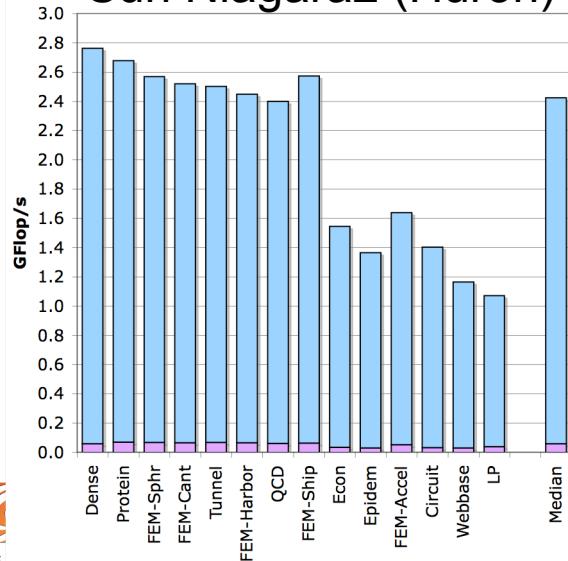
Intel Clovertown



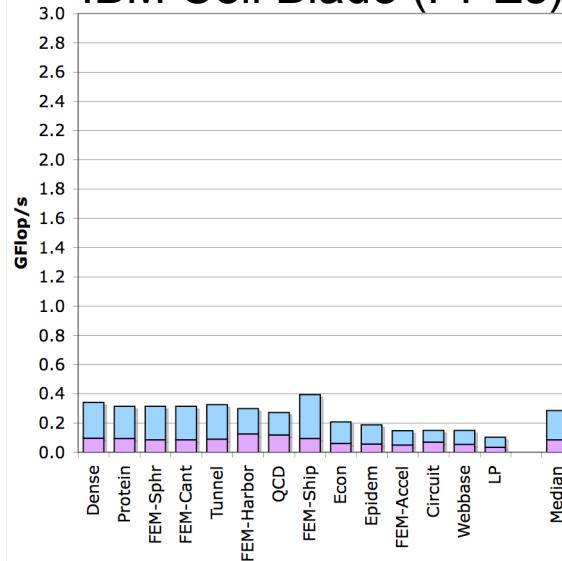
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



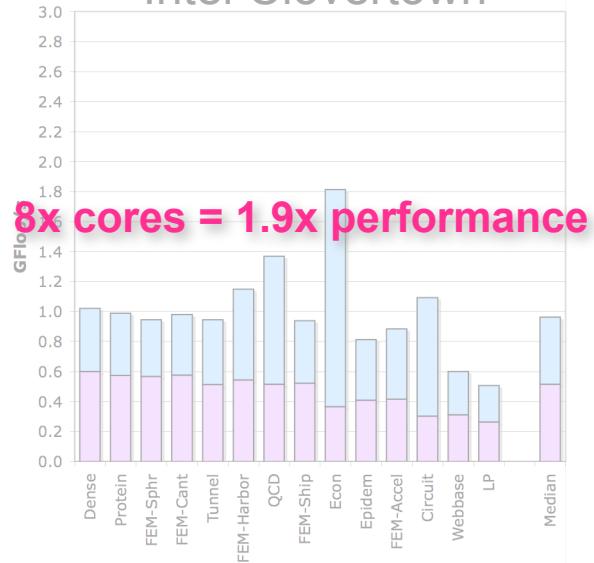
- ❖ SPMD style
- ❖ Partition by rows
- ❖ Load balance by nonzeros
- ❖ N2 ~ 2.5x x86 machine

- Naive Pthreads
- Naive

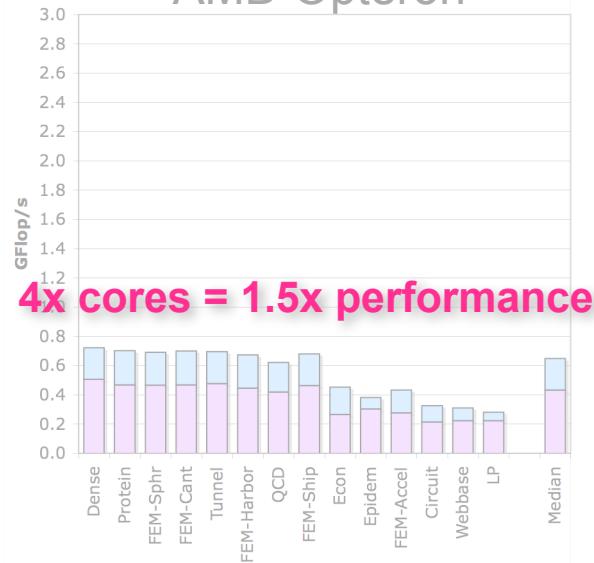
Naïve Parallel Implementation



Intel Clovertown

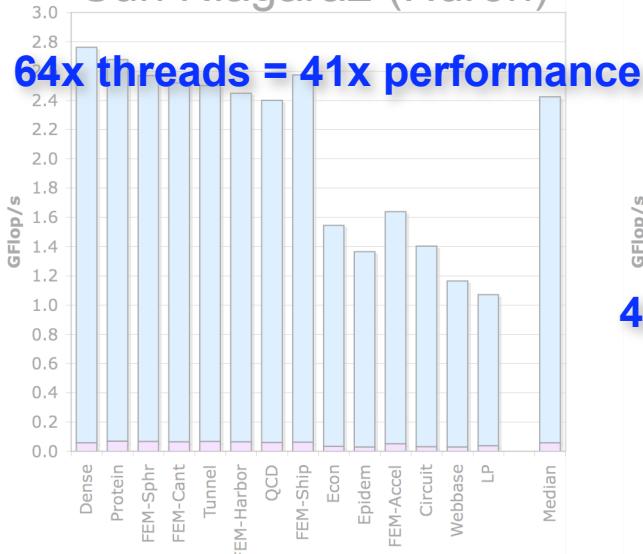


AMD Opteron

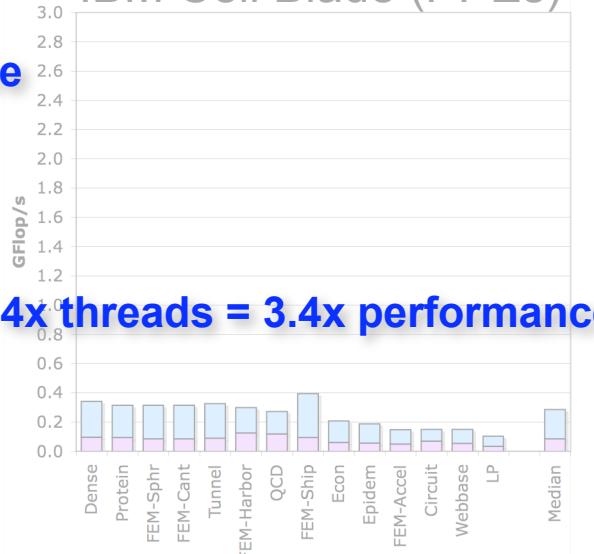


- ❖ SPMD style
- ❖ Partition by rows
- ❖ Load balance by nonzeros
- ❖ N2 ~ 2.5x x86 machine

Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)

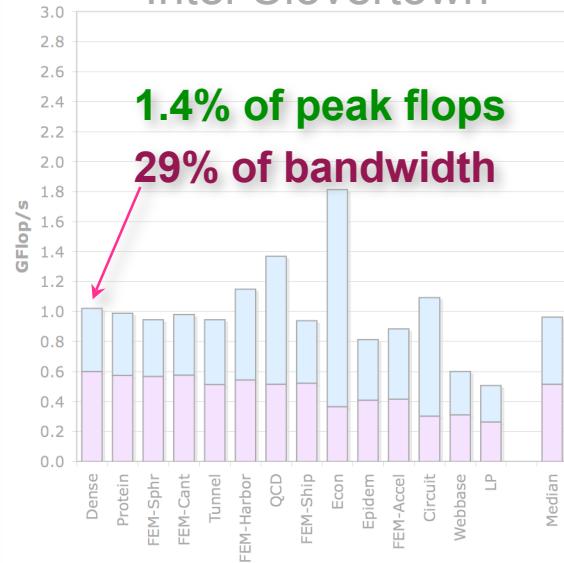


- ❖ Naïve Pthreads
- ❖ Naïve

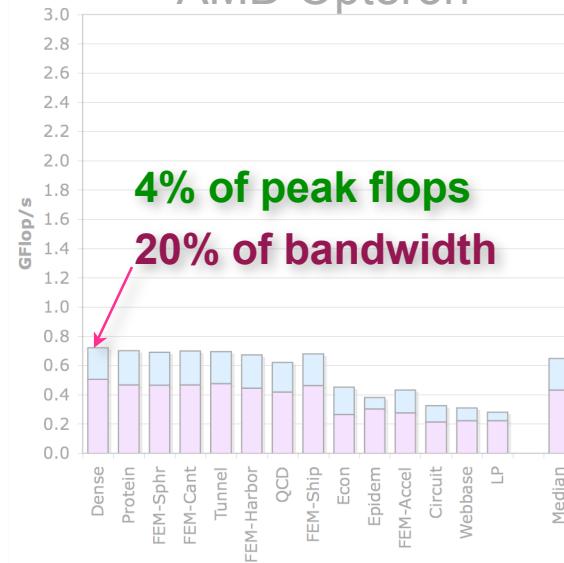
Naïve Parallel Implementation



Intel Clovertown

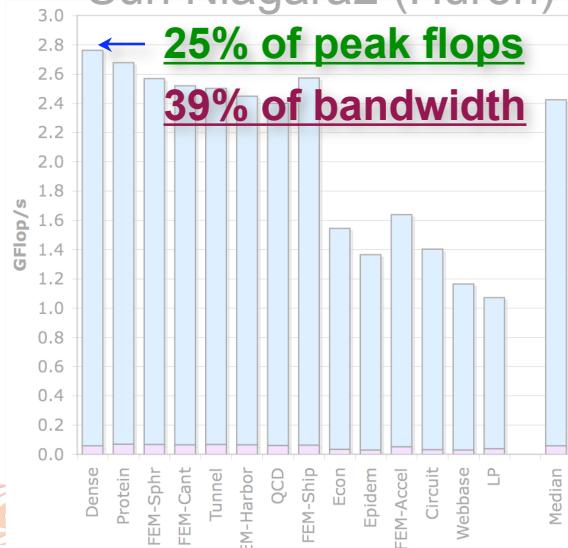


AMD Opteron

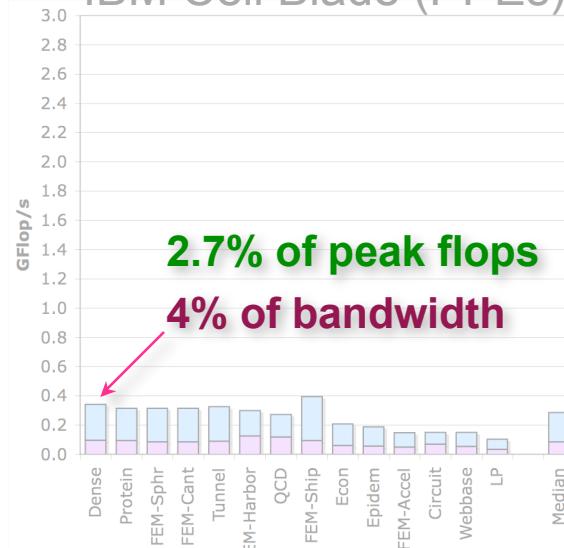


- ❖ SPMD style
- ❖ Partition by rows
- ❖ Load balance by nonzeros
- ❖ N2 ~ 2.5x x86 machine

Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- ❖ Naïve Pthreads
- ❖ Naïve

Case for Autotuning



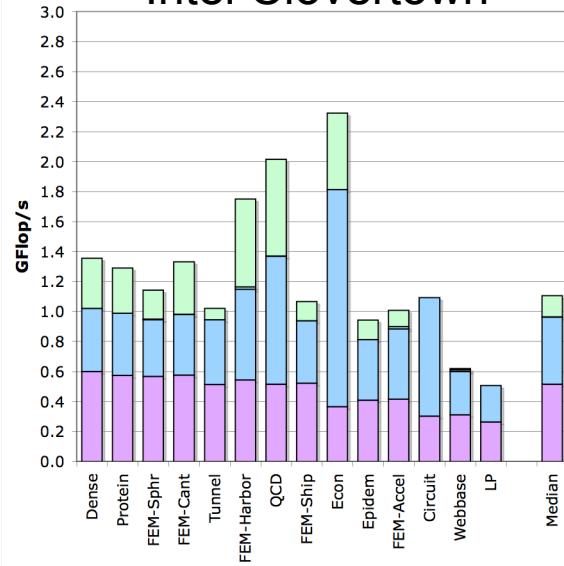
- ❖ How do we deliver good performance across all these architectures across all matrices?
- ❖ None obvious choices for data structures can yield highest efficiency
- ❖ Autotuning (examples: ATLAS, FFTW)
 - Perl script can generates all possible optimizations
 - Heuristically, or exhaustively search the optimizations
 - Existing SpMV solution: OSKI (developed at UCB)
- ❖ This work:
 - Optimizations geared for multi-core/-threading
 - generates SSE/SIMD intrinsics, prefetching, blocking loop transformations (SW pipelining), alternate data structures, etc...

Autotuned Performance

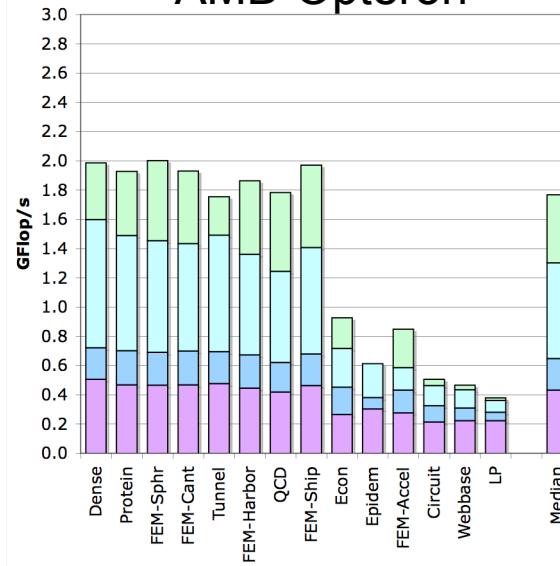
(+NUMA & SW Prefetching)



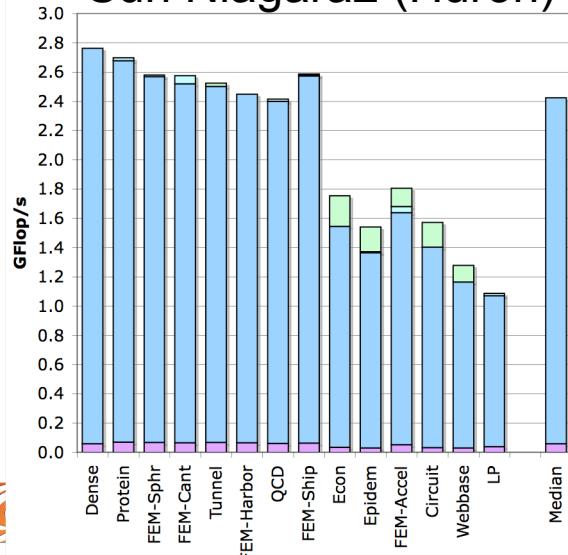
Intel Clovertown



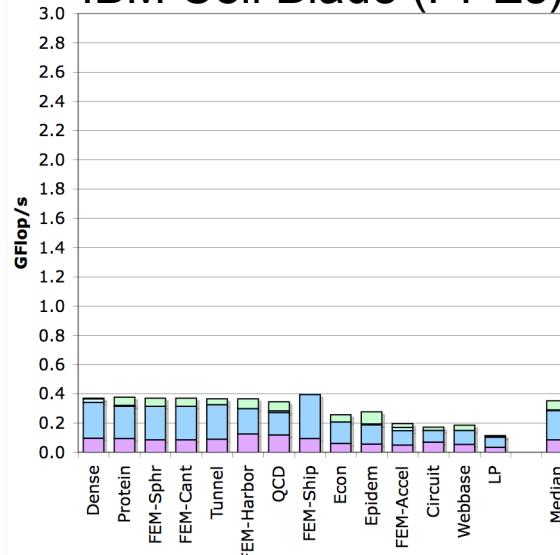
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)

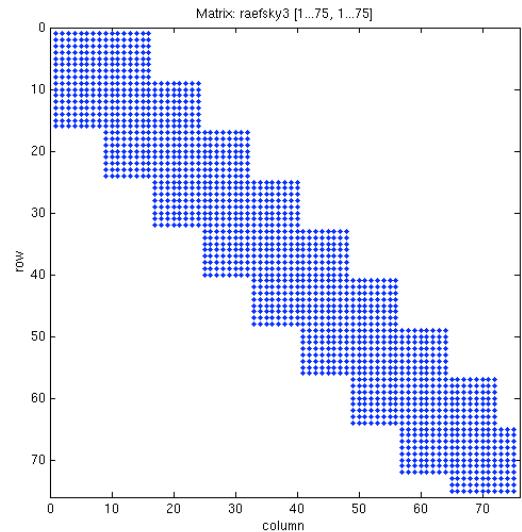


- ❖ Use first touch, or libnuma to exploit NUMA.
- ❖ Includes process affinity.
- ❖ Tag prefetches with temporal locality
- ❖ **Autotune:** search for the optimal prefetch distances

Matrix Compression



- ❖ For memory bound kernels, minimizing memory traffic should maximize performance
- ❖ Compress the meta data
 - Exploit structure to eliminate meta data
- ❖ **Heuristic:** select the compression that minimizes the matrix size:
 - power of 2 register blocking
 - CSR/COO format
 - 16b/32b indices
 - etc...
- ❖ **Side effect:** matrix may be minimized to the point where it fits entirely in cache

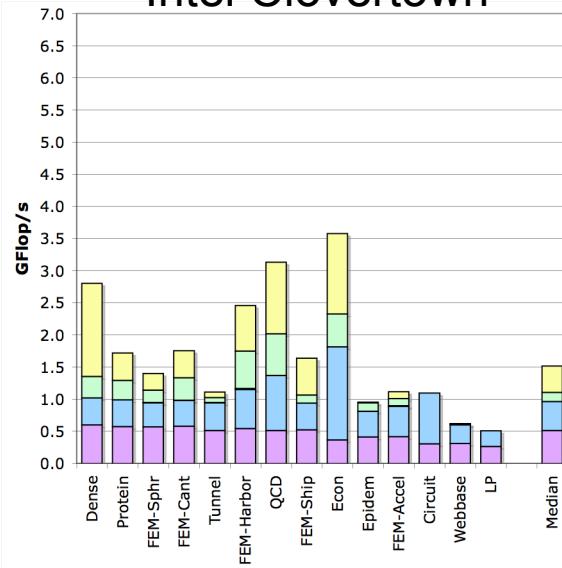


Autotuned Performance

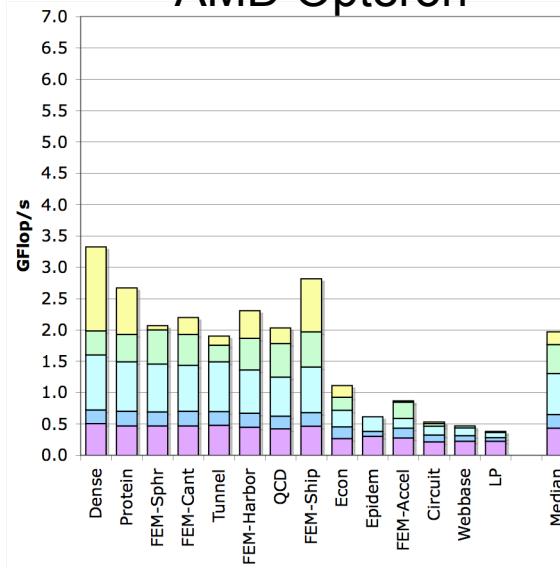
(+Matrix Compression)



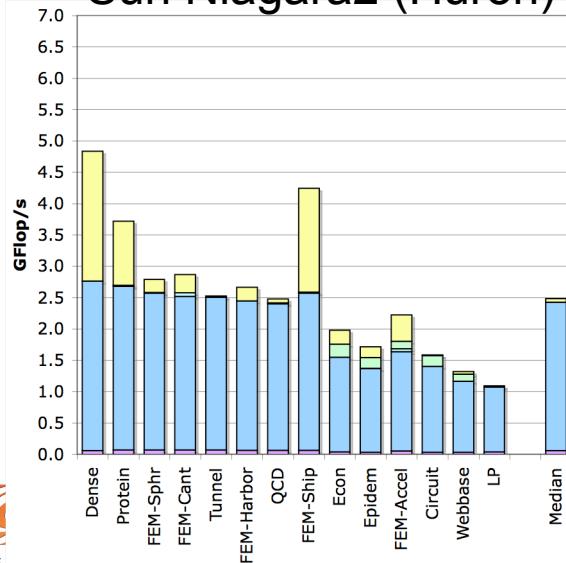
Intel Clovertown



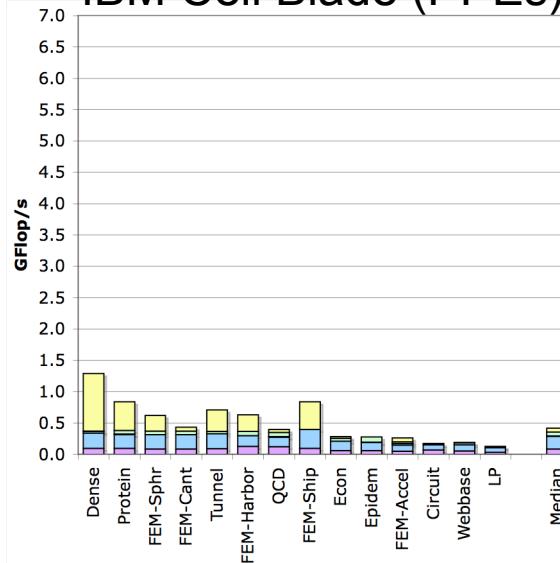
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- ❖ If memory bound, only hope is minimizing memory traffic
- ❖ Heuristically compress parallelized matrix to minimize it
- ❖ Implemented with SSE
- ❖ Benefit of prefetching is hidden by requirement of register blocking
- ❖ Options: register blocking, index size, format, etc...

- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Cache and TLB Blocking

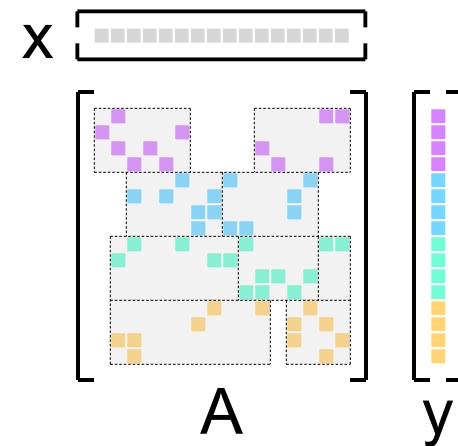


- ❖ Accesses to the matrix and destination vector are streaming
- ❖ But, access to the source vector can be random
- ❖ Reorganize matrix (and thus access pattern) to maximize reuse.
- ❖ Applies equally to TLB blocking (caching PTEs)

- ❖ **Heuristic:** block destination, then keep adding more columns as long as the number of source vector cache lines(or pages) touched is less than the cache(or TLB). Apply all previous optimizations individually to each cache block.

- ❖ **Search:** neither, cache, cache&TLB

- ❖ Better locality at the expense of confusing the hardware prefetchers.

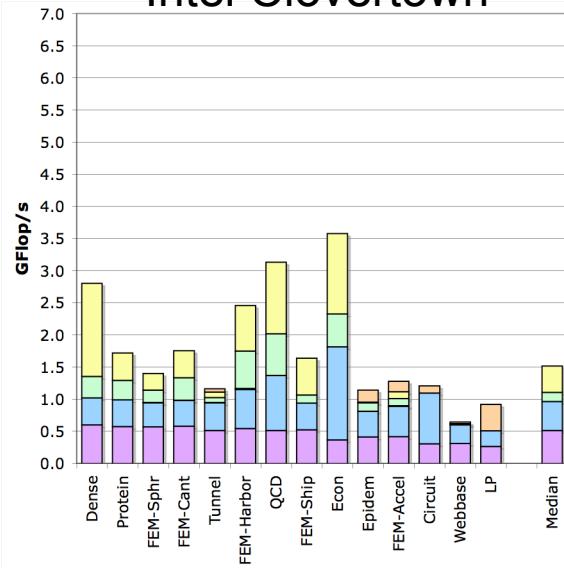


Autotuned Performance

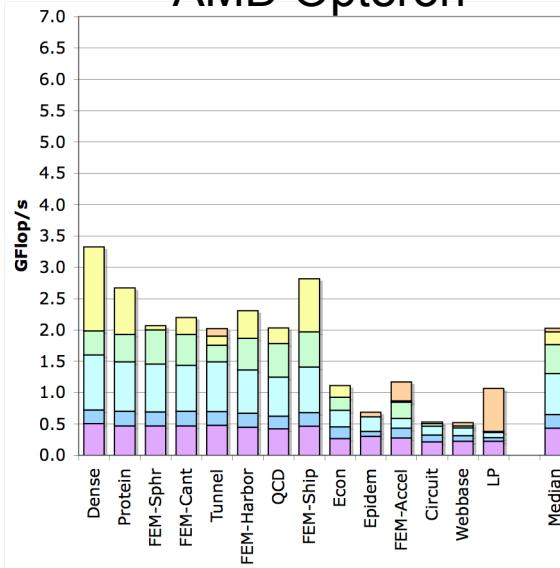
(+Cache/TLB Blocking)



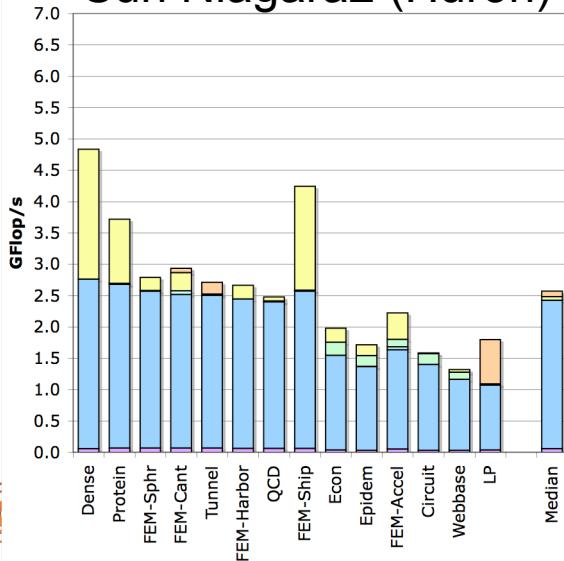
Intel Clovertown



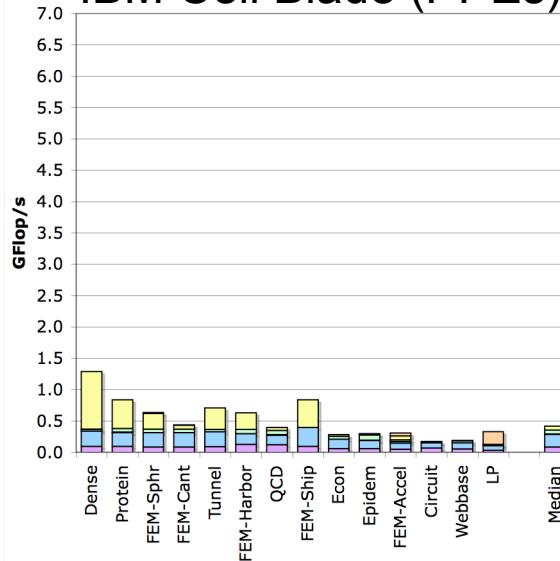
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- ❖ Reorganize matrix to maximize locality of source vector accesses

- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Banks, Ranks, and DIMMs



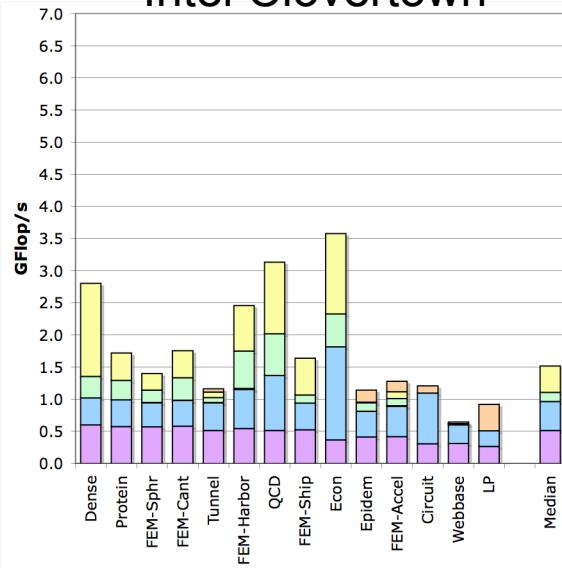
- ❖ In this SPMD approach, as the number of threads increases, so to does the number of concurrent streams to memory.
 - ❖ Most memory controllers have finite capability to reorder the requests. (DMA can avoid or minimize this)
 - ❖ Addressing/Bank conflicts become increasingly likely
-
- ❖ Add more DIMMs, configuration of ranks can help
 - ❖ Clovertown system was already fully populated

Autotuned Performance

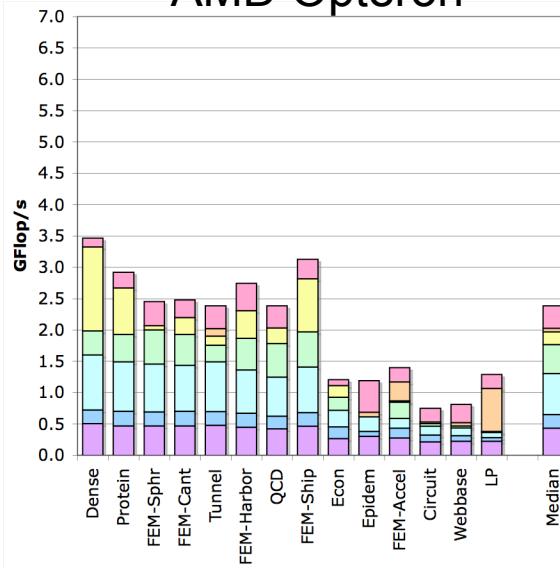
(+DIMMs, Firmware, Padding)



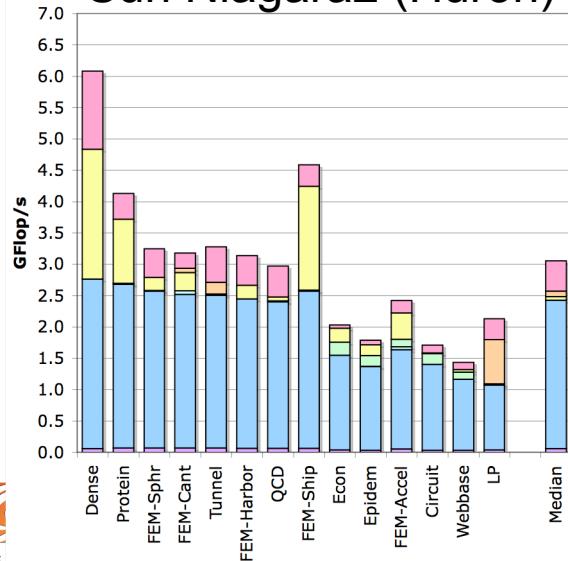
Intel Clovertown



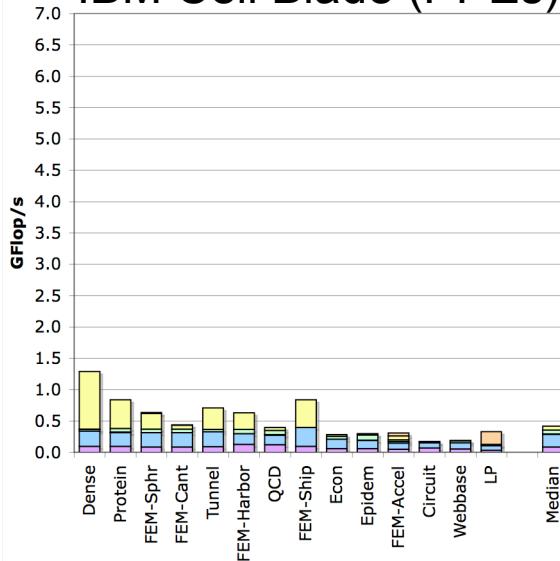
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- ❖ Clovertown was already fully populated with DIMMs
- ❖ Gave Opteron as many DIMMs as Clovertown
- ❖ Firmware update for Niagara2
- ❖ Array padding to avoid inter-thread conflict misses

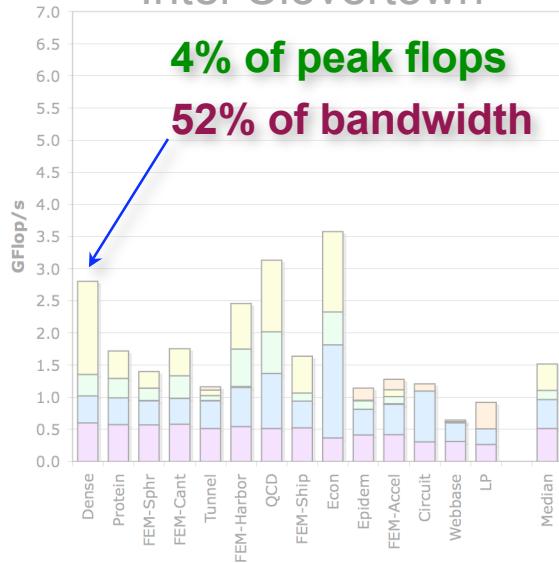
- +More DIMMs(Opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Autotuned Performance

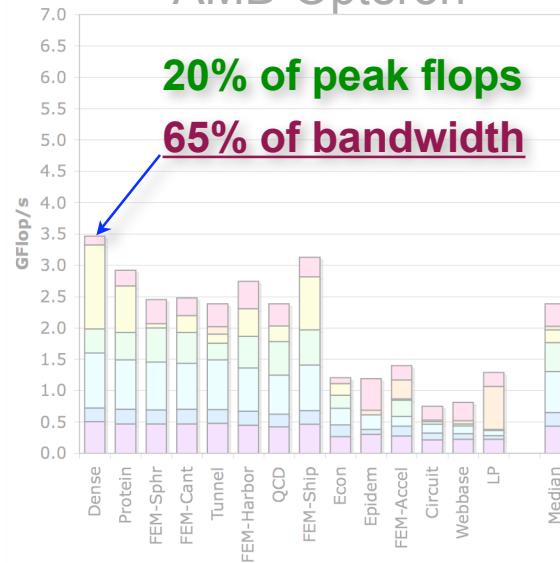
(+DIMMs, Firmware, Padding)



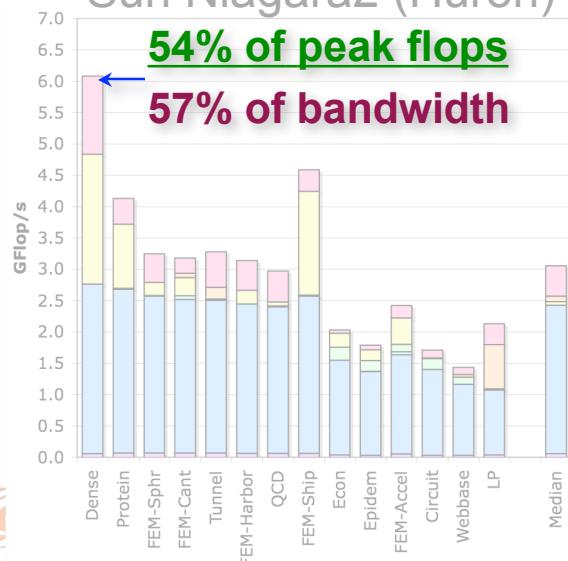
Intel Clovertown



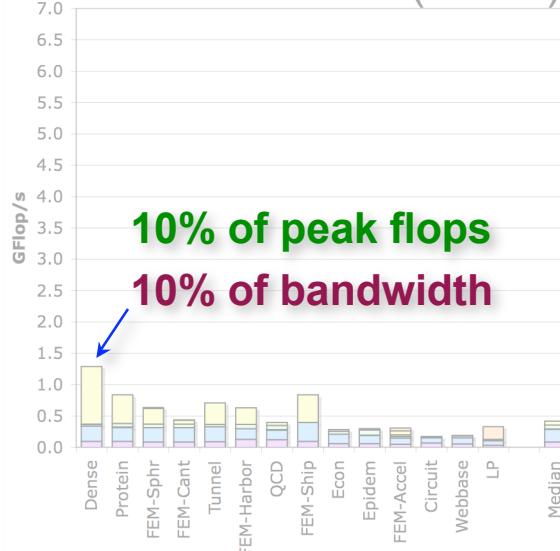
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)



- ❖ Clovertown was already fully populated with DIMMs
- ❖ Gave Opteron as many DIMMs as Clovertown
- ❖ Firmware update for Niagara2
- ❖ Array padding to avoid inter-thread conflict misses
- ❖ PPE's use ~1/3 of Cell chip area

- +More DIMMs(opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Cell Implementation



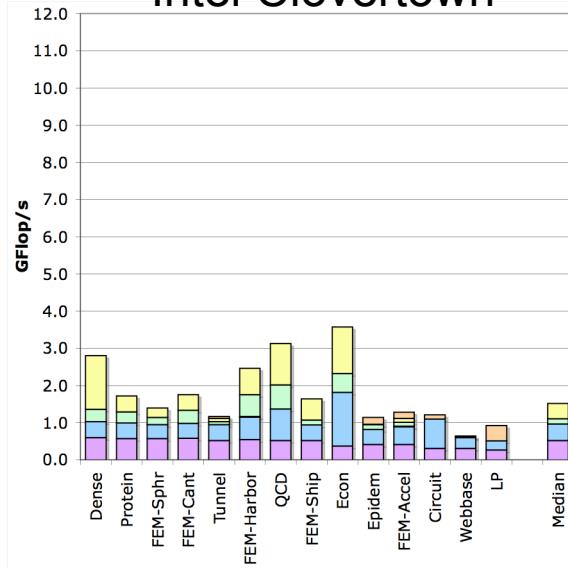
- ❖ **No vanilla C implementation (aside from the PPE)**
- ❖ Even SIMDized double precision is extremely weak
 - Scalar double precision is unbearable
 - Minimum register blocking is 2x1 (SIMDizable)
 - **Can increase memory traffic by 66%**
- ❖ Cache blocking optimization is transformed into local store blocking
 - **Spatial and temporal locality is captured by software when the matrix is optimized**
- ❖ No branch prediction
 - Replace branches with conditional operations
- ❖ In some cases, what were optional optimizations on cache based machines, are requirements for correctness on Cell
- ❖ Despite the performance, Cell is still handicapped by double precision

Autotuned Performance

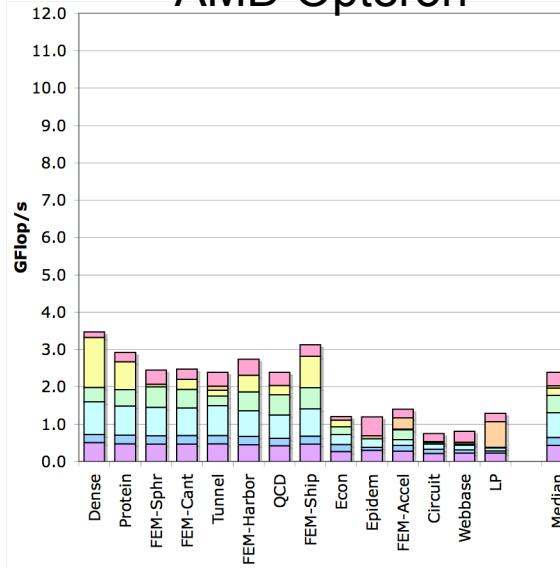
(+Cell/SPE version)



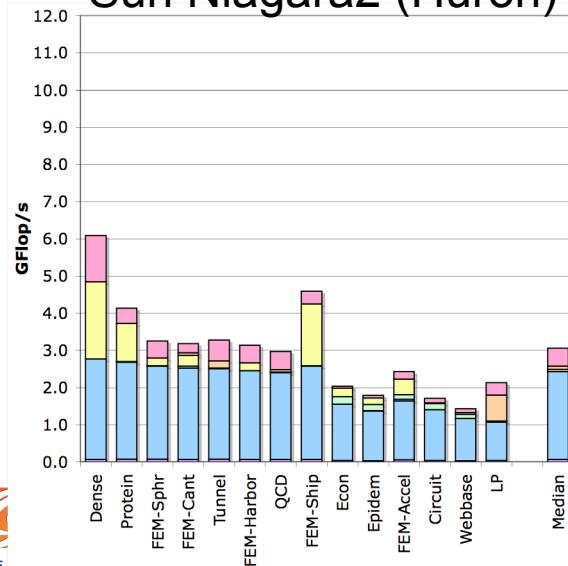
Intel Clovertown



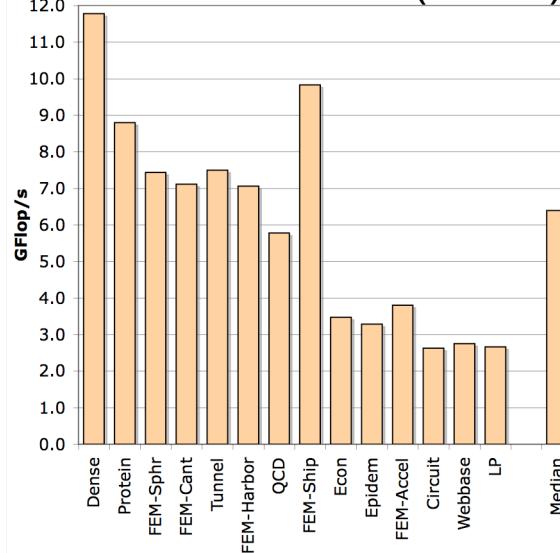
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (SPEs)



- ❖ Double precision Cell Version
- ❖ DMA, local store blocked, NUMA aware, etc...
- ❖ Only 2x1 and larger BCOO
- ❖ About 12x faster (median) than PPEs alone.

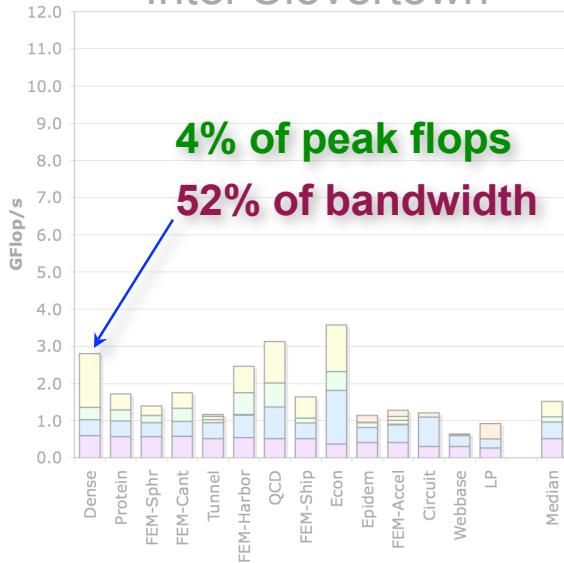
- +More DIMMs(opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Autotuned Performance

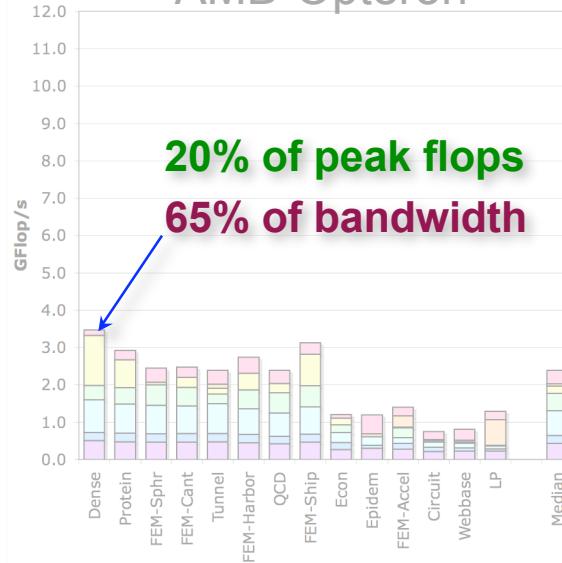
(+Cell/SPE version)



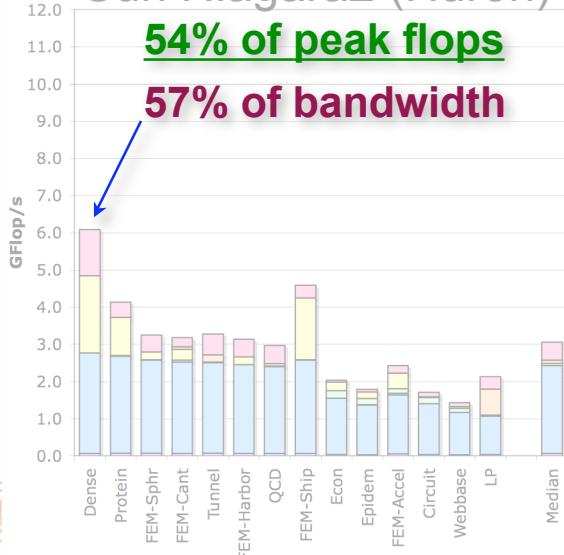
Intel Clovertown



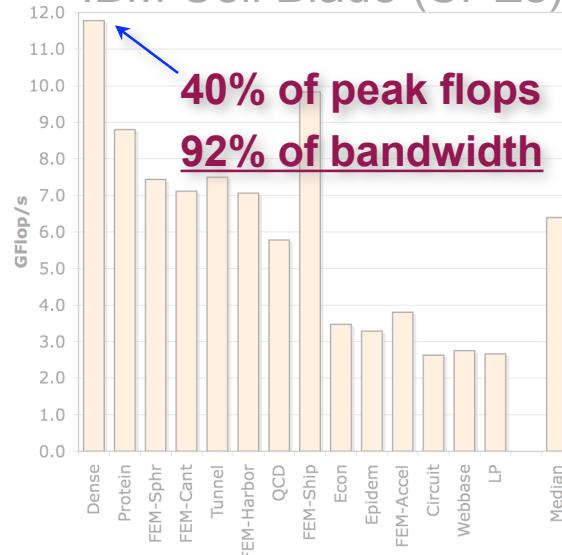
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (SPEs)



- ❖ Wrote a double precision Cell/SPE version
- ❖ DMA, local store blocked, NUMA aware, etc...
- ❖ Only 2x1 and larger BCOO
- ❖ Only the SpMV-proper routine changed
- ❖ About 12x faster than using the PPEs alone.

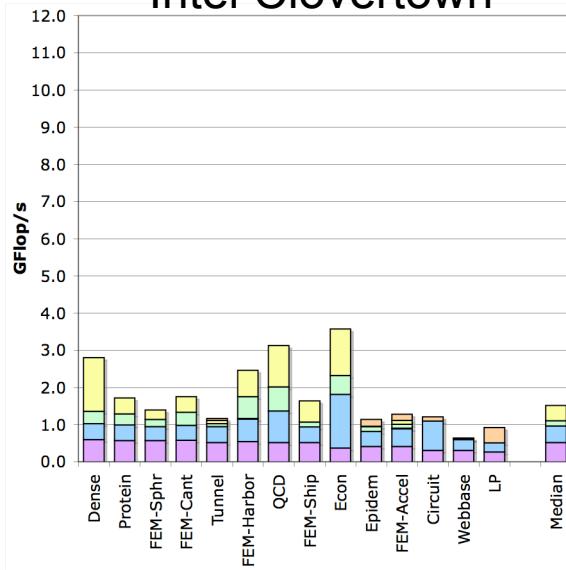
- +More DIMMs(opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

Autotuned Performance

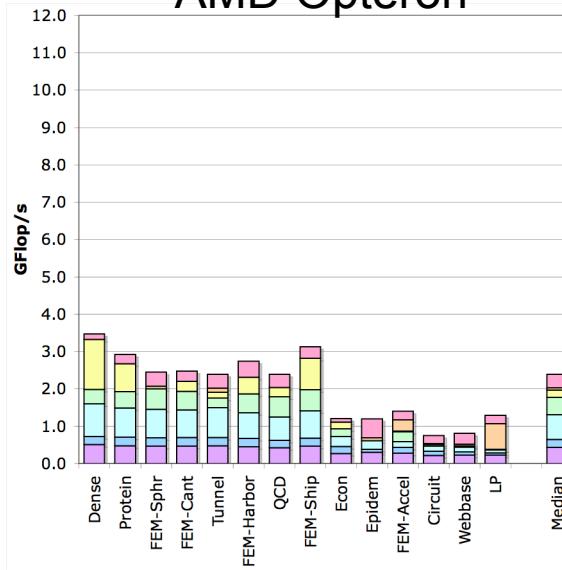
(How much did double precision and 2x1 blocking hurt)



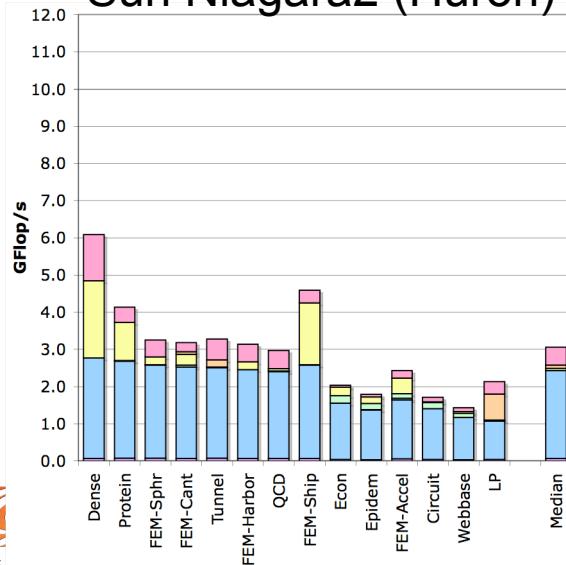
Intel Clovertown



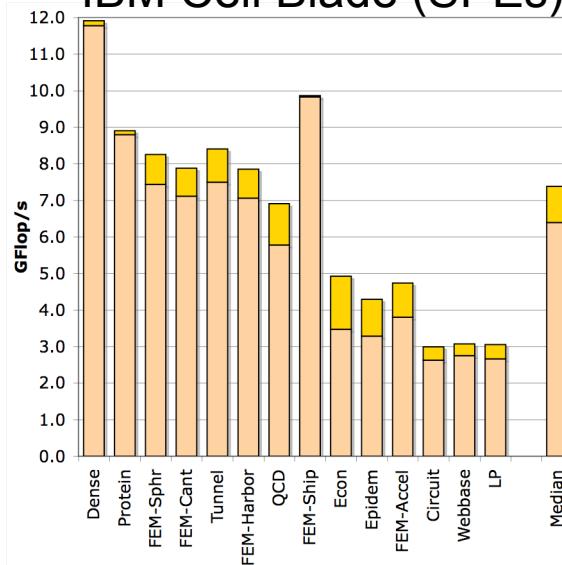
AMD Opteron



Sun Niagara2 (Huron)



IBM Cell Blade (SPEs)



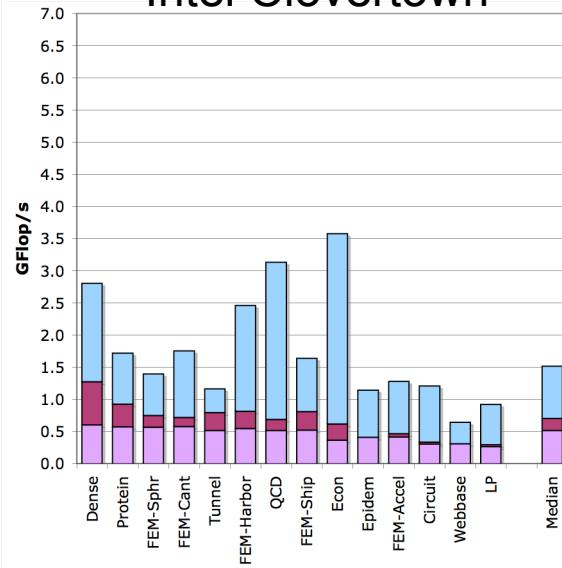
- ❖ Model faster cores by commenting out the inner kernel calls, but still performing all DMAs
- ❖ Enabled 1x1 BCOO
- ❖ ~16% improvement

- +better Cell implementation
- +More DIMMs(opteron), +FW fix, array padding(N2), etc...
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

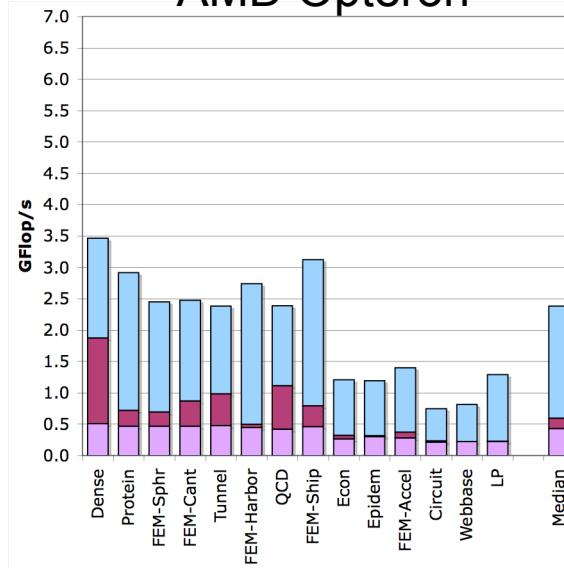


MPI vs. Threads

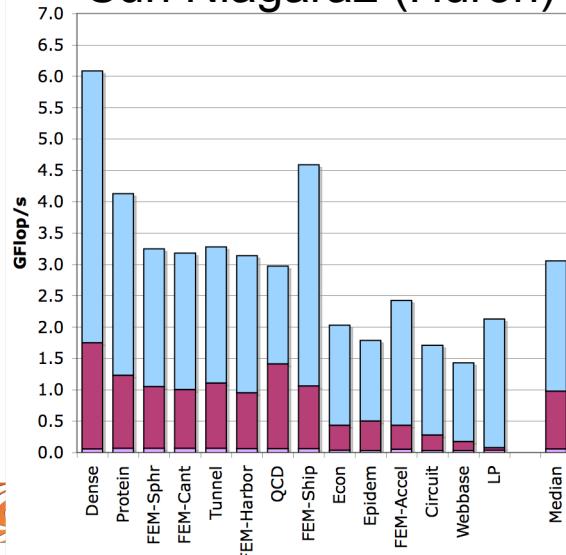
Intel Clovertown



AMD Opteron



Sun Niagara2 (Huron)



- Autotuned pthreads
- Autotuned MPI
- Naïve Serial

- ❖ On x86 machines, autotuned(OSKI) shared memory MPICH implementation rarely scales beyond 2 threads
- ❖ Still debugging MPI issues on Niagara2, but so far, it rarely scales beyond 8 threads.



C O M P U T A T I O N A L R E S E A R C H D I V I S I O N

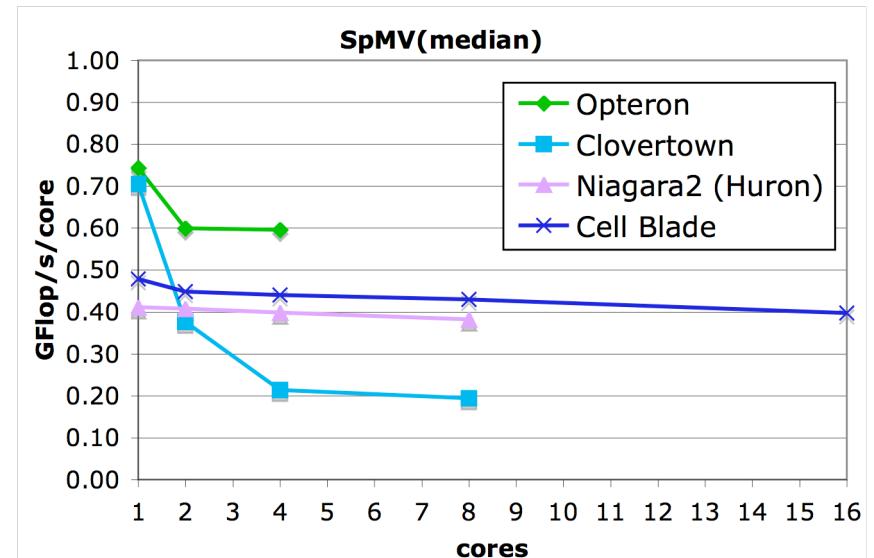
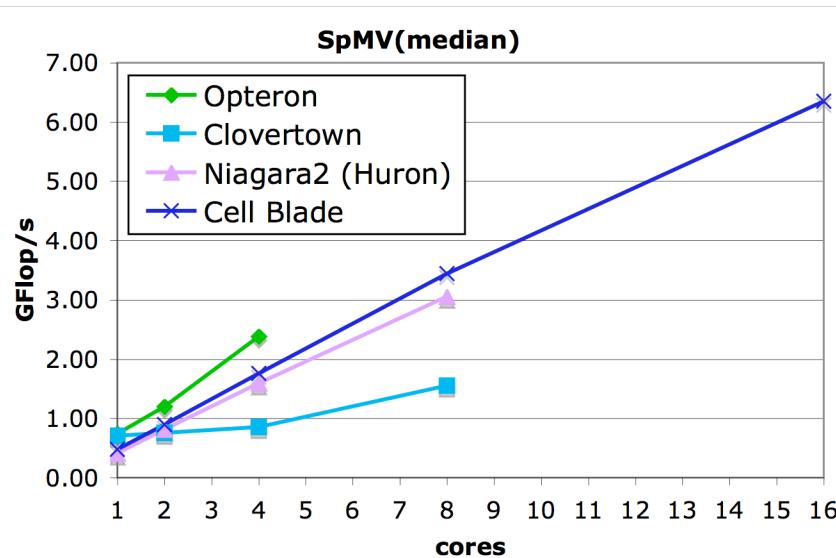
Summary

Performance & Efficiency

(Fully optimized)



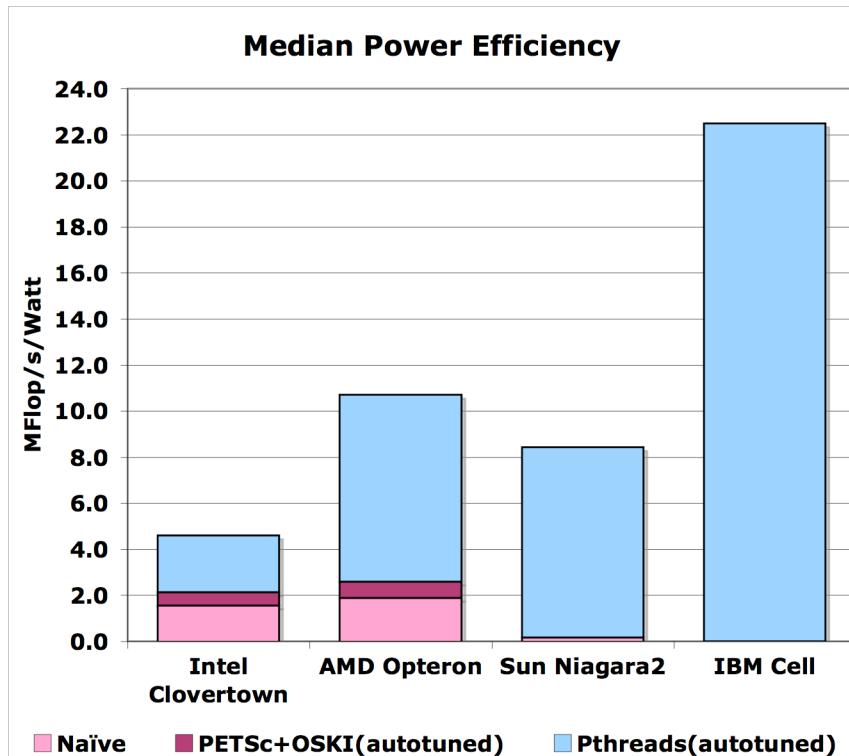
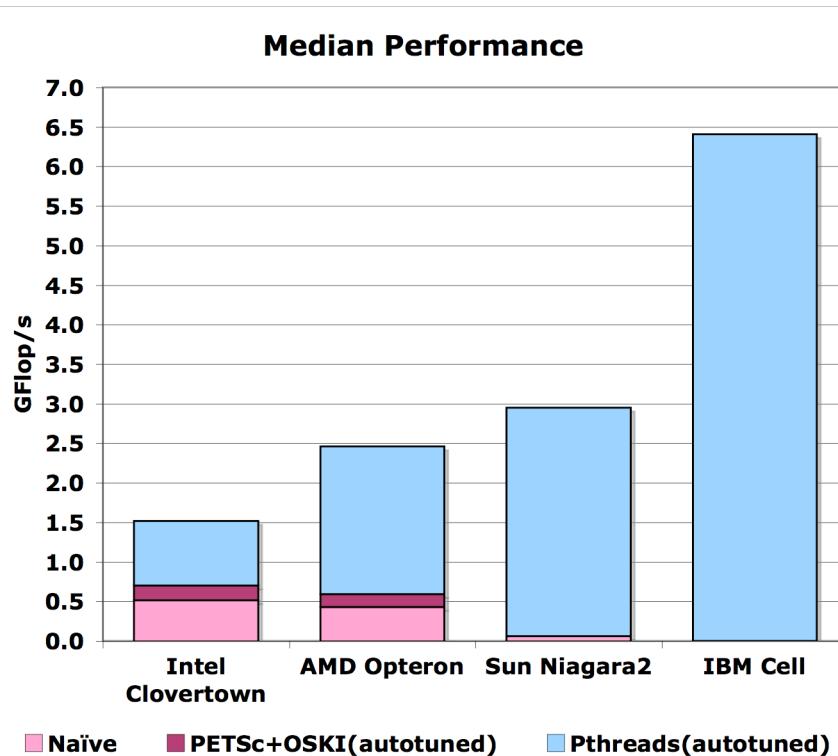
- ❖ Cell consistently delivers the best full system performance
 - Niagara2 delivers comparable per socket performance
- ❖ Niagara2 has far more bandwidth than it can exploit (too much latency, too few cores)
- ❖ Dual core Opteron delivers far better performance (bandwidth) than Clovertown, but as the flop:byte ratio increases its performance advantage decreases.
 - Even though peak Clovertown 4.2X Opteron
- ❖ Clovertown has far too little effective FSB bandwidth
 - Showed poor multicore scaling



Median Performance & Efficiency



- ❖ Used digital power meter to measure sustained system power
 - FBDIMM drives up Clovertown and Niagara2 power
 - 8 DIMMs on Clovertown (~330W), 16 DIMMs on N2 (~450W)
 - Opteron (~300W), Cell (~285W)
- ❖ Default approach(MPI) achieves very low performance and efficiency



Summary



- ❖ **Paradoxically**, the most complex/advanced architectures required the most tuning, and delivered the lowest performance.
- ❖ Most machines achieved less than 50-60% of DRAM bandwidth
- ❖ Niagara2 delivered both good performance and productivity
- ❖ Cell delivered very good performance and efficiency
 - 90% of mem BW, high power efficiency, easily understood performance
- ❖ **Multicore specific autotuned implementation significantly outperformed a state of the art MPI implementation**
 - ✓ Multi-core optimized: Matrix compression, NUMA, Prefetching
- ❖ Results suggest that CMPs designs should maximize effective bandwidth utilization with increasing cores, even at the cost of single core performance

Acknowledgements



- ❖ UC Berkeley
 - RADLab Cluster (Opterons)
 - PSI cluster(Clovertowns)
- ❖ Sun Microsystems
 - Niagara2 access
- ❖ Forschungszentrum Jülich
 - Cell blade cluster access